UNIT - IV

VISIBLE SURFACE DETERMINATION

Definition

* Given a collection of 3D objects and a 2D viewing specification, the process of determining the lines or surfaces of 3D objects that are visible and displaying only those surfaces or lines are visible is called as visible line or surface determination.

* Process is also called as hidden line elimination or ~~visible~~ hidden surface elimination algorithm.

Visible surface determination algorithm works on two approaches

1) object space method
2) Image space method.

1) ~~obm~~ object space method :- visible surface determination is done on physical co-ordinates before projection.

2*) Image space method:-

Visible surface determination is done on screen co-ordinates after projection

imp
Techniques for efficient visible surface determination :-

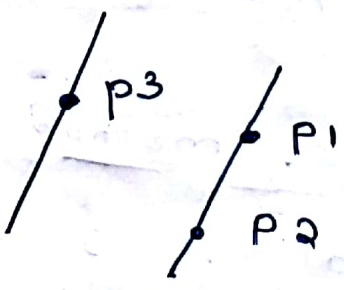
- Decreasing the overhead of the algorithm;

1*) Perspective transformation :-

→ It is used to reduce the overhead involved in perspective projection visible surface determination.

→ The technique aims on finding the visibility of the surface before it is projected on to the screen as after projection looses some information on the depth.

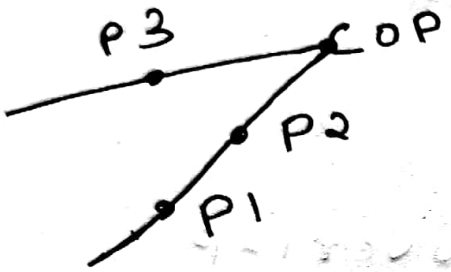
Parallel projection :-



condition :-

p1 (x1, y1, z1)
p2 (x2, y2, z2)
x1 = = x2 &
y1 = = y2

perspective :-



condition :- $\frac{x1}{z1} = = \frac{x2}{z2}$ && $\frac{y1}{z1} = = \frac{y2}{z2}$

A 3D object is initially transformed sueh o way that parallel projection of transformed object is equals to the perspective projection of untransformed object.

ii) Extents and Bounding Box :-

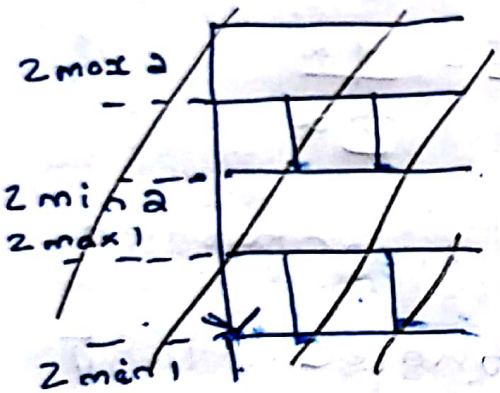
* This technique aims in reduction of unnecessary comparisons between the object for the visible surface determination.

~~* Extra Extents are~~

* Extents are the rectangular surface surrounding the projection of 3D object.

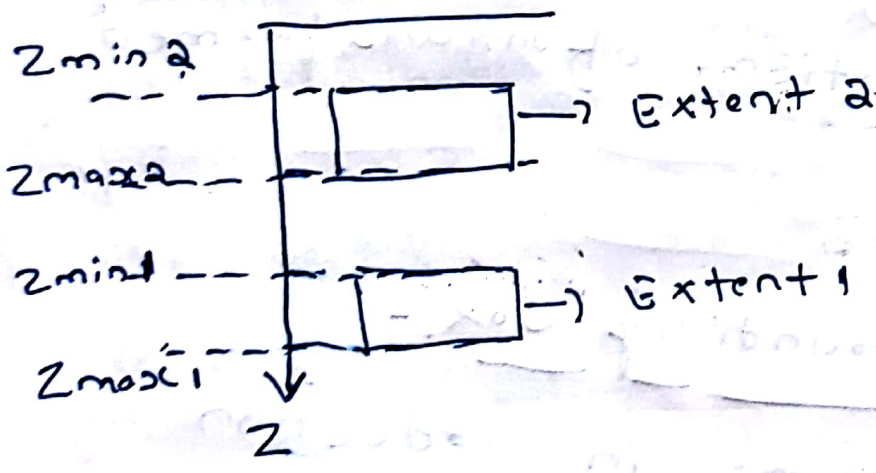
(If 2 extent ~~does~~ doesnot overlap then it will never overlap)

(But if they overlap then it may or may not overlap in future)



no overlap if

Zmax 2 < Zmin 1
or
Zmax 1 < Zmin 2



Bounding Volume (Box) :- (Before projection)

Rectangular surface enclosing object itself is called as bounding volume or a

bounding Box

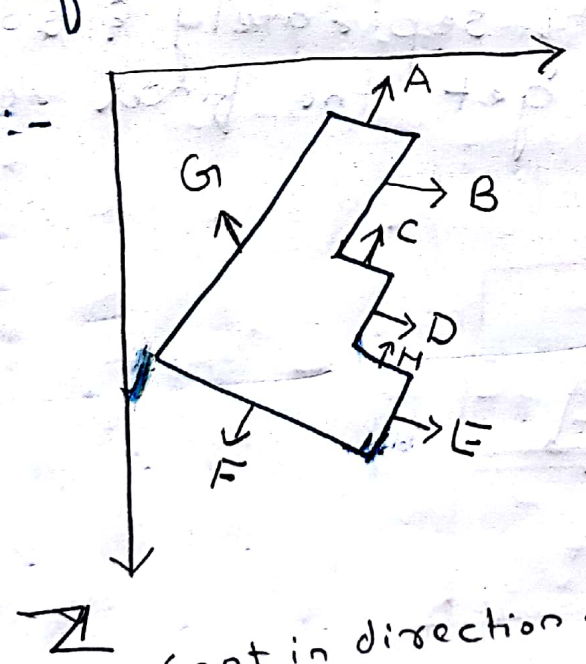
iii) Back face ~~coding~~ culling (cutting)

* A back face culling technique is applicable when the object is approximated as ~~polygon~~ polyhedron whose interior is not exposed to the front side.

(polyhedron :- made up of n no. of polygon surfaces)

* The technique ~~identifys~~ identifies those polygons whose surface normal points away from the observer and eliminates the surfaces out of the visible surface determination.

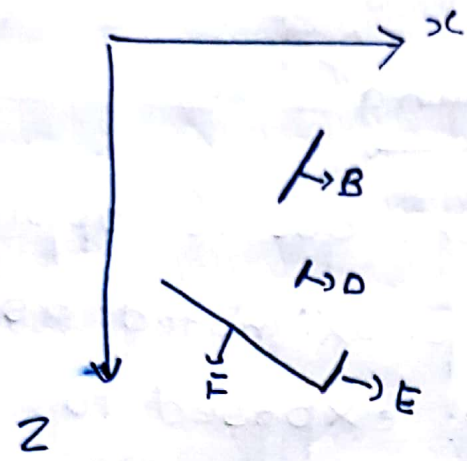
Ex :-



~~eliminate~~

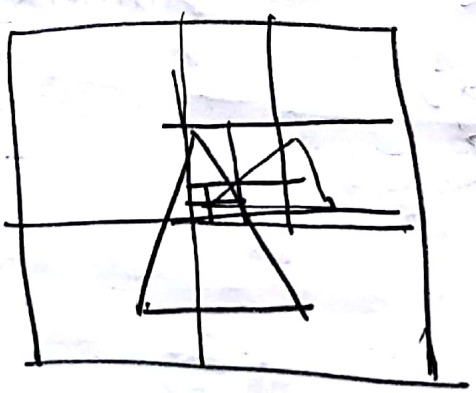
eliminate : A, C, H, G (not in direction of z)

noteliminate : B, D, E, F



iv) Spatial Partitioning

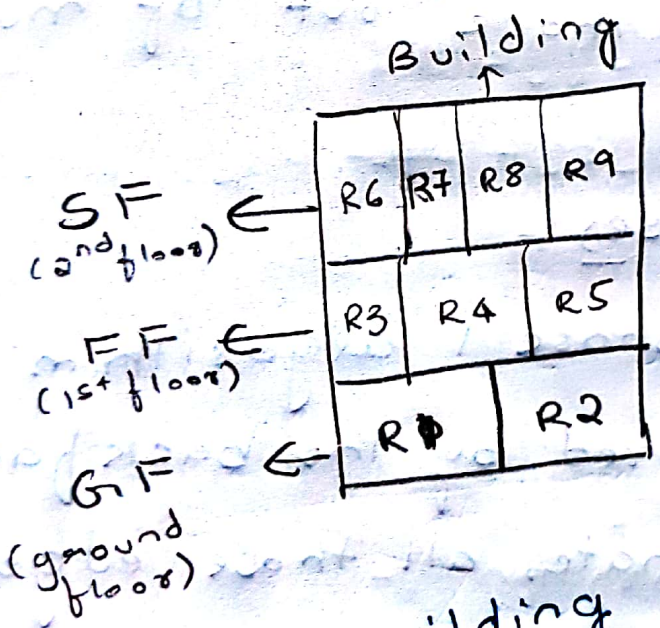
* It is based on the divide and conquer strategy in which an area sub-division rule is applied to divide an entire area into smaller sub problems.
* Each area is ~~extraire~~ examined to find out the visibility seperately & solutions are combined to get the final result.

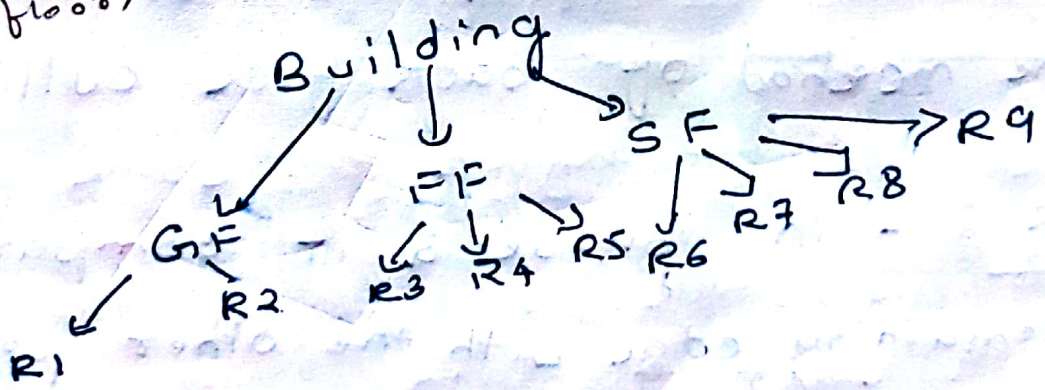


v) ~~Hierarchy~~ : Hierarchy :-

* A problem is basically ~~re~~ realized in terms of hierarchical tree structure.

* In computer graphics an object given that needs to be project is first realised as heirarchical tree structure & then visibility is determined individually in each hierarchy.





Algorithms for visible Line determination

* Robert's Algorithm.
* Appel's Algorithm.
* Haloed lines.

* Robert's Algorithm:-

→ Invented by Robert's

→ Algorithm is only applicable for the convex polyhedron made out of multiple polygon surface.

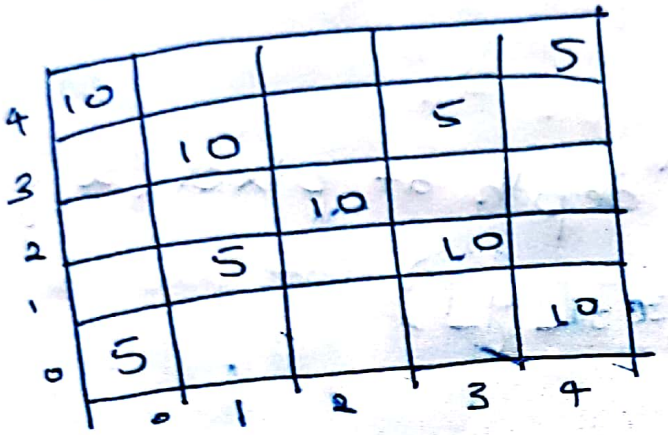
Works in 2 phases:-

phase 1:- In phase 1 algorithm identifies the edges shared by the back facing polygon and eliminates all those edges by the method of back face culling.

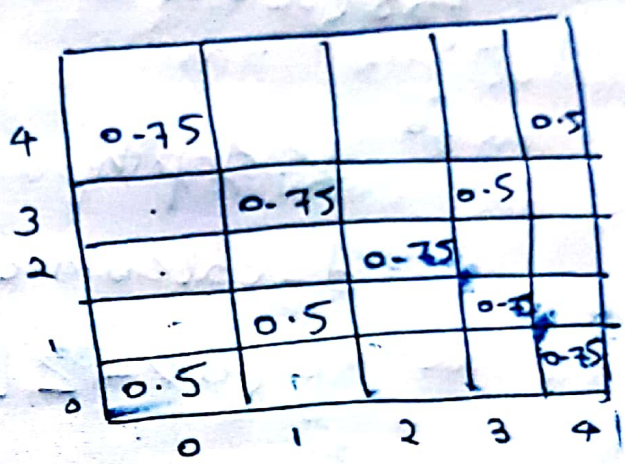
phase 2:- In phase 2 it compares each remaining edge with the other polyhedron to determine the visibility.

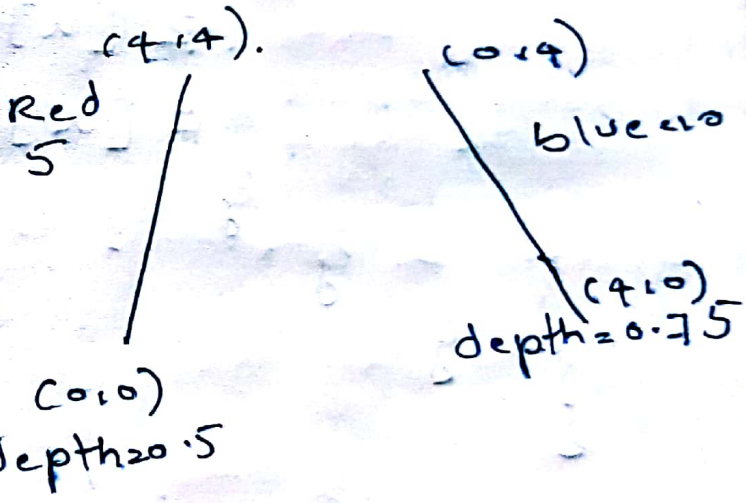
Z Buffer Algorithm (Depth Buffer Algorithm)

Frame - buffer



Z - buffer





Algorithm :-

```
for ( y=0 ; y < ymaxc ; y++ )
{
    for (x=0; x < xmaxc ; x++)
    {
        Z[x][y] = 0;
        I=[x][y] = BG_COLOR;
    }
}
```

```
for (each polygon in the list)
{
  for (y = ymin, y <= ymax; y++)
  {
    for (x = xmin; x <= xmax; x++)
    {
      d = depth
      d = calculate depth at pixel (x, y);
      if (d >= Z[x][y])
      {
        Z[x][y] = d;
        F[x][y] = poly POLYGON_COLOR;
      }
    }
  }
}
```

List priority algorithm :-

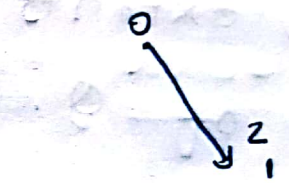
Create a list of objects on some priority criteria.

* Depth Sort Algorith.
* Binary Space Partitioning Tree Algorith (BSP).

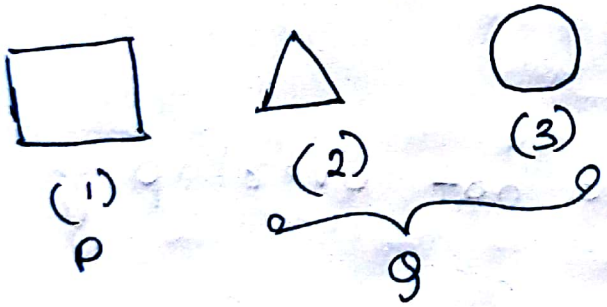
* Depth sort Algorithm :- (Painter's Algorithm)

* It is an algorithm in which objects are given the priority based on the depth value..

* Farther objects are given highest priority than the closer objects. i.e the objects are sorted initially based on the decreasing depth value i.e they are sorted on the lowest z-co-ordinate value.



(O is given 1st priority, 1 is given last priority)



Algorithm is also called as Painter's Algorithm

Conceptual steps :-

1) * Sort all the polygons based on the increasing

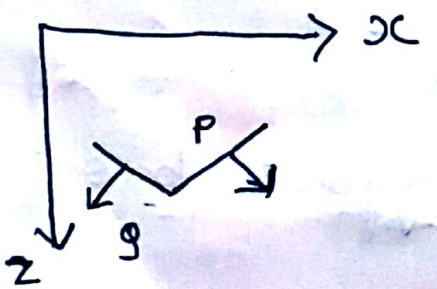
value of z co-ordinate.

(2)* For the selected polygon P from the list check if P obscures any Q remaining in the list and if P obscures Q then resolve the ambiguity. else goto to step 3

(3)* Scan convert polygon P.

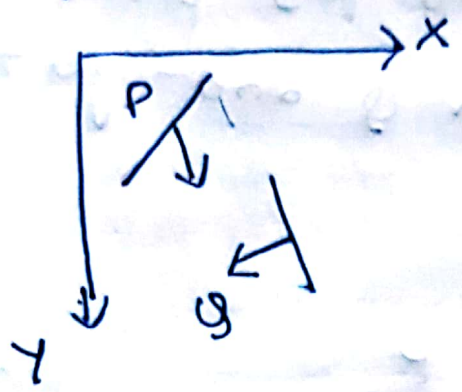
Sequential test to prove P does not obscure Q

1. If z extents of P do not overlap z extents of Q.
2. If x extents of P do not overlap x extent of Q.
3. If y extents of P do not overlap y extents of Q.
4. If P is completely on the opposite side of Q's plane as view point.



(Q is on backside of P)

5. If Q is completely on the same side of P's plane as viewpoint.



(Q is on front side of P)

Algorithm :-

Depth Sort Algorithm :-

1. Sort the list of polygon based on the smallest z co-ordinate value and create a sorted list called polylist.
2. Select & ~~first polygon~~ remove the first polygon from the polylist and name it as P.
3. for each polygon Q in polylist do the following steps :-
   i) Check if every Q passes test 1 to 5 with respect to P.

ii) If any Q fails the test then swap P & Q and split polygon Q into Q1 and Q2 ~~and~~ replace w.r.t to P's plane and replace Q1 & Q2 with Q and go to step 1 else goto step (iii)

(iii) Paint the polygon P

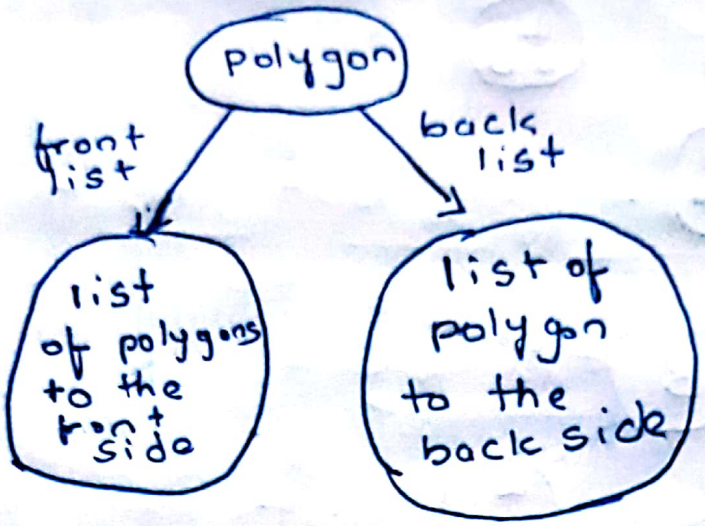
4) Goto step 2.

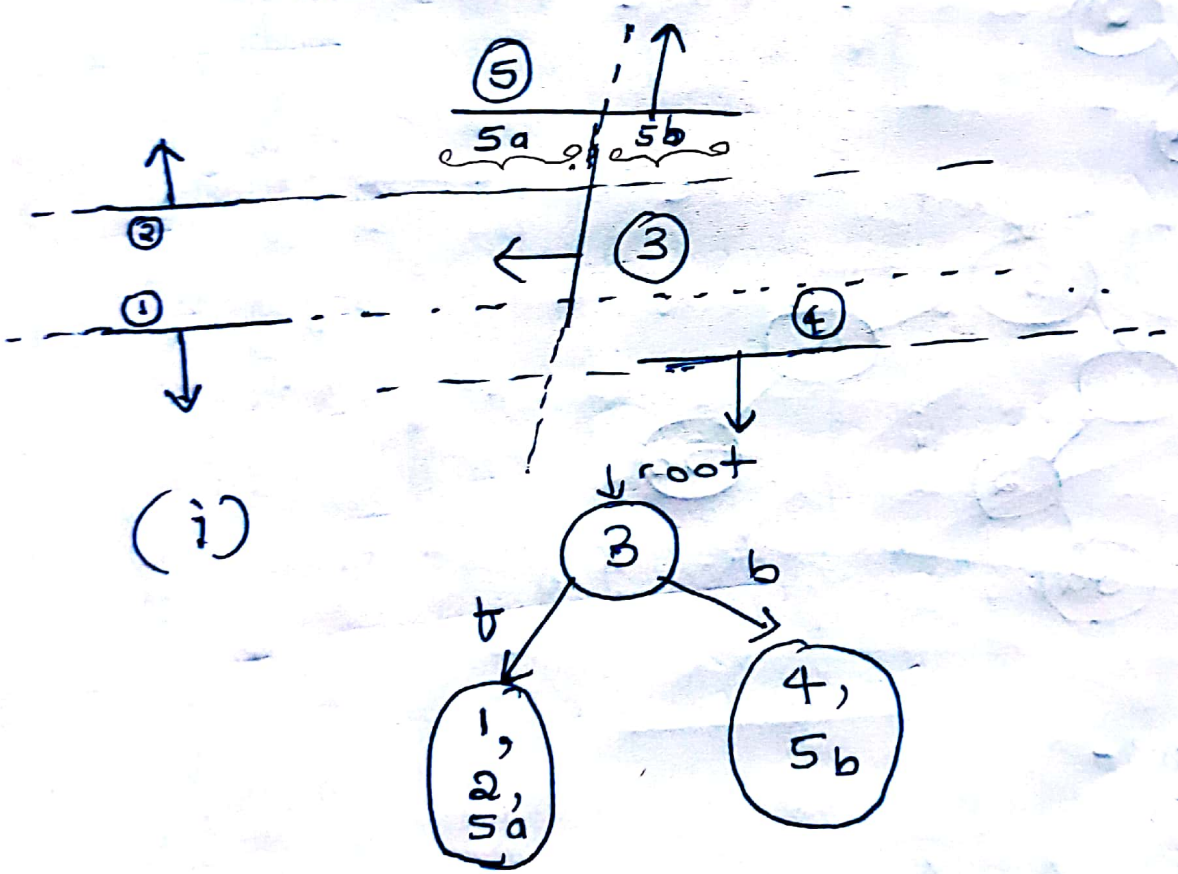
Binary Space Partitioning (BSP) Tree Algorithm

* In this algorithm List is created based on the binary tree structure.

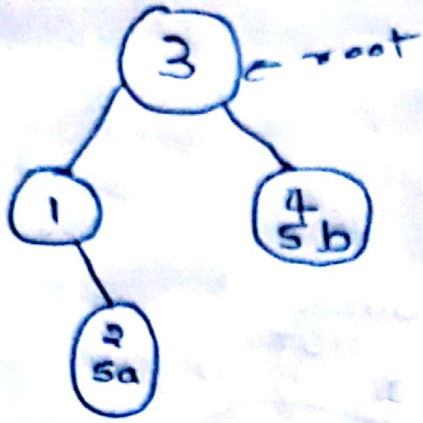
* This algo does not require any pre sorting process & tree structure is created arbitrarily from particular polygon.

* This algorithm is event useful in creating a display from any view point.

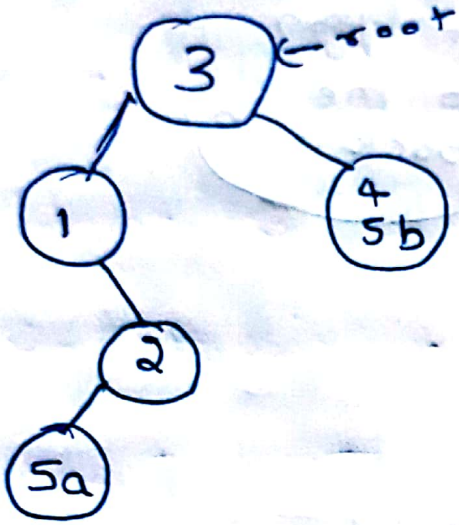




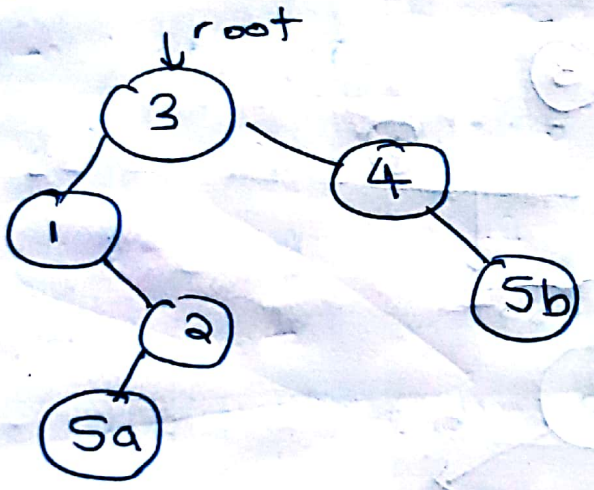
(ii)



(iii)



(iv)



Algorithm :-

i) Select any polygon as a root node.

ii)

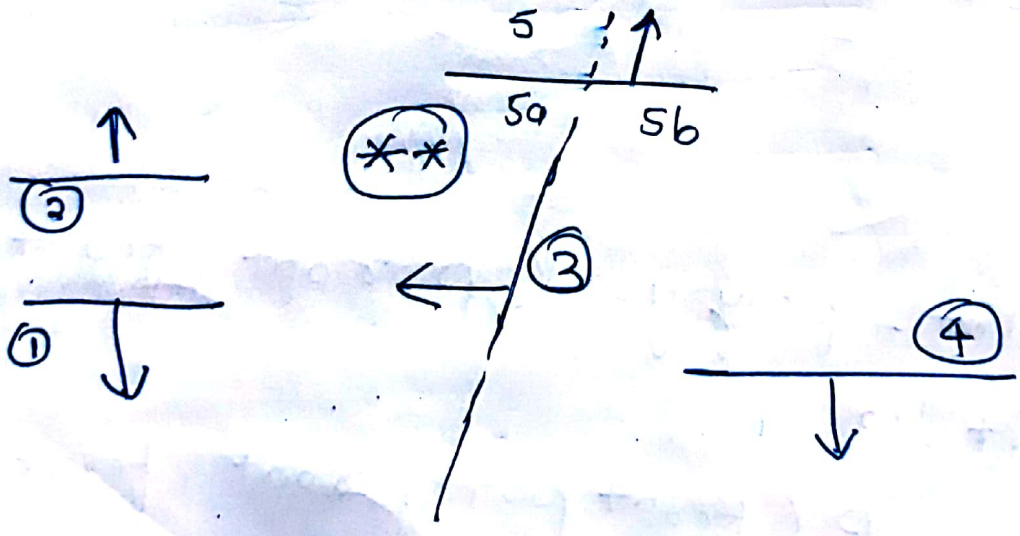
```
create_BSP_TREE(polylist)
{
  if(root == NULL)
  {
    return
  }
root = Select and remove a polygon from
polylist.
for(each remaining polygon p in the polylist)
{
  if(p in front of root)
  {
    add_BSP(p, root, frontlist);
  }
  else if(p back of root)
  {
    add_BSP(p, root, backlist);
  }
  else
  {
    split_polygon(p, root, frontpart,
backpart);
    add_BSP(frontpart, root, frontlist);
    add_BSP(backpart, root, backlist);
}}
```

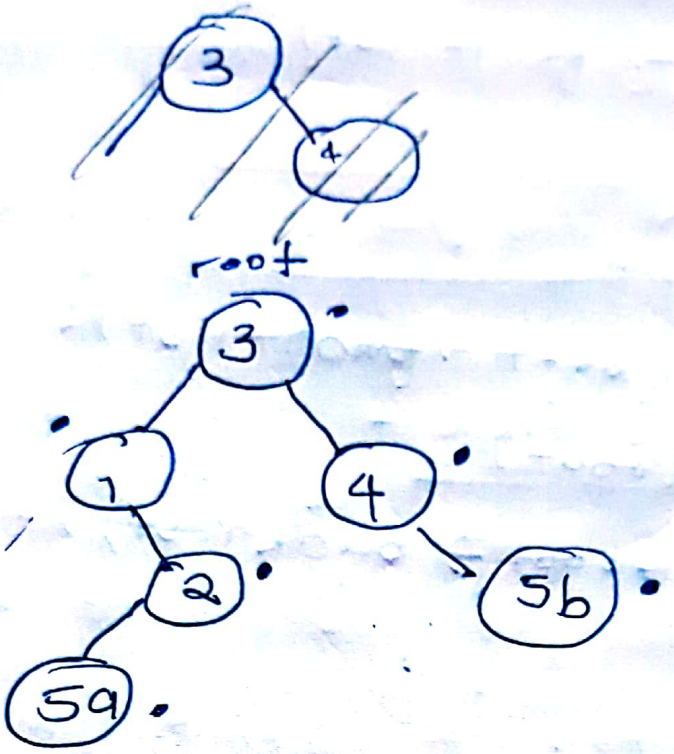
```
create_BSP_TREE(frontlist);
create_BSP_TREE(backlist);
}
```

Displaying list from BSP tree :-

Recursive procedure :-

* If viewpoint is in front of root
  - (i) Recursively trace root → backchild
  - (ii) visit root
  - (iii) Recursively trace root → front child
* If viewpoint is towards back of root.
  - i) recursively trace root → front child
  - ii) Visit root
  - iii) Recursively trace root → basic child





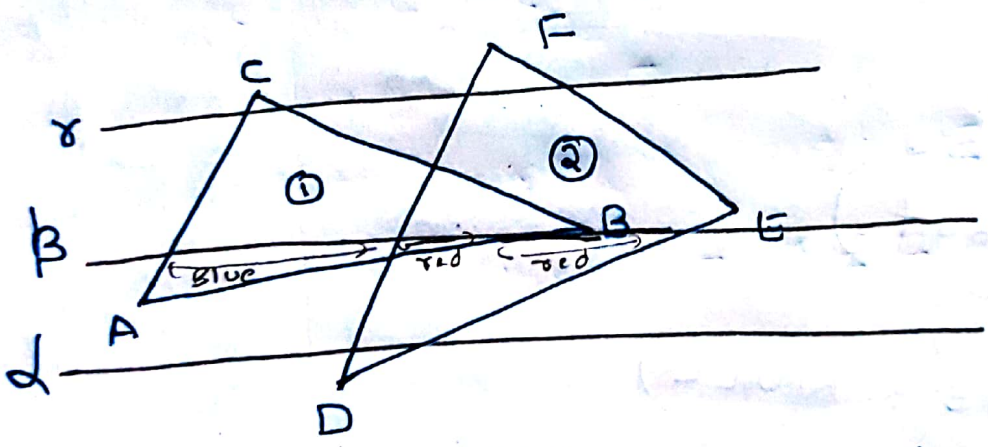
4, 5b, 3, 1, 2, 5a

Algorithm :-

```
display (root)
{
  if (root = = NULL)
  {
    return ;
  }
  if (v in front of root)
  {
    display (root -> backlist);
```

```
        visit(root);
    display(root -> frontlist);
    }
    else
    {
        display (root -> frontlist);
        visit (root);
        display (root -> backlist);
    }
}
```

Scanline Algorithm



* Algorithm is similar to ~~scalin~~ scanline Algorithm for polygon filling.
* Deals with multiple polygon surfaces.
* This algorithm uses ~~tree~~ three Data structures
  i) Edge table.

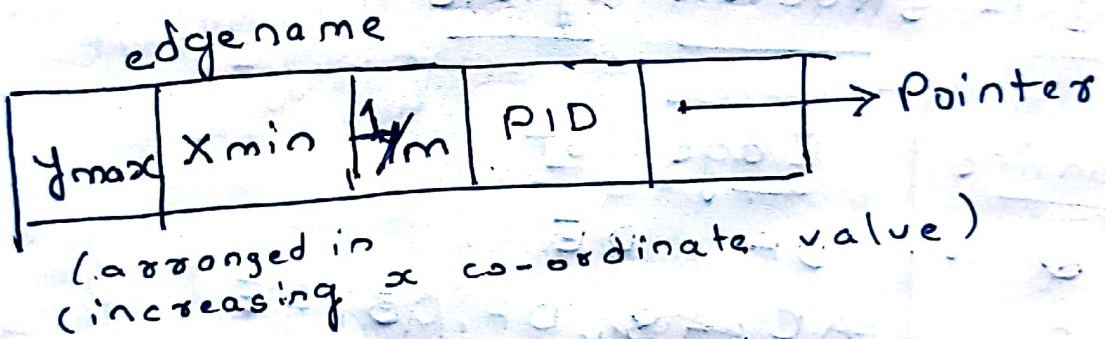
ii) Active edge table.

iii) Polygon surface table.

* Edge table:-

→ This table stores all the information regarding the edges that are intersected by each scanline y = ymin to ymax.

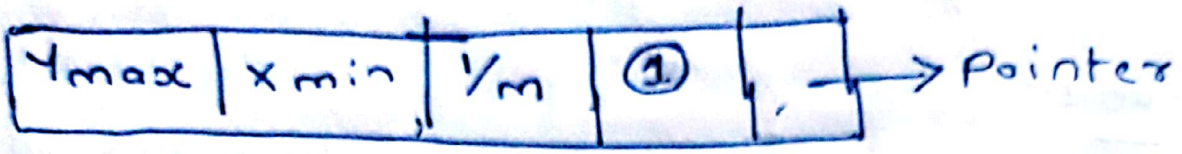
→ Each edge of information is stored in the form of node having 5 different fields.



ymax - largest y co-ordinate
xmin - smallest x "
m - slope
PID - id of the polygon surface to which the edge belongs to.
Pointer - point to next scanline.

for ex :-

~~edge~~ To have information of edge AC



2) Active Edge table :-

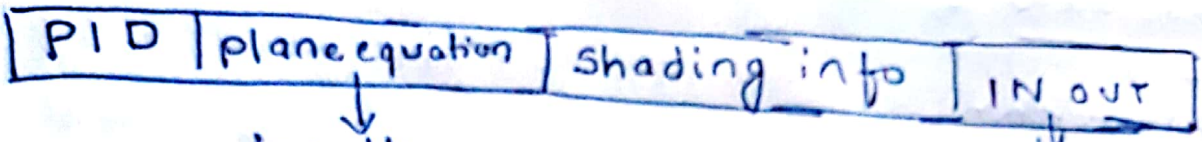
It is a table that will have list of edges activated for particular scanline that ranges from $y = y_{min}$ to y_{max}

Ex :- Structure of AET :-

Scanline	Edge list
α	DF , DE
β	AC ~~AB~~ DF, AB, DE
γ	AC, BC, DF, ~~EF~~ EF

3) Polygon surface table :-

* It is a table that keeps track of the information of each polygon surface and it also tells if a particular surface is activated for particular scanline.



Boolean field which specifies if the surface is active or not.
if it is 0 surface is not active
if 1 it is active.

Polygon surface table:-

1	PE1	Blue	0
2	PE2	Red	0

Assume Polygon surface ② is front of ①

For scanline 2 it is completely Red.

Procedure :-

Algorithm :-

1. Initialize ET
2. Initialise polygon surface table.
3. $y = y_{min}$ smallest y co-ordinate value.
4. For $y = y_{min}$ to y_{max} do the following.
   (i) $x = x_{min}$ of intersecting edge.
   (ii) invert in-out flag corresponding to the intersected edge.
   (iii) If $y = y_{max}$ of any edge then delete the edge from AET
   (iv) Count = number of in-out flags that are 1
   (v) If (count $\geq$ 1) then find the closest surface and color the pixel range with shade of the closest surface. else
   (vi) goto (vi)
   (vi) Shade the range of pixel with the color of Activated surface.

(vii) $x_2 = x + 1/m$

(viii) $y_2 = y + 1$

Area Subdivision Algorithm

* Works based on the rule of divide & conquer strategy used in spatial ~~partioating~~ partitioning.
* It basically tries to make the decision on the area of projection & if the decision couldnot be made then the area is recursively divided into smaller part.

2 Algorithms :-

* Warnock's Algorithm.
* ~~Weiler Ar~~
* Weiler Atherton Algorithm.

* Warnock's Algorithm :-

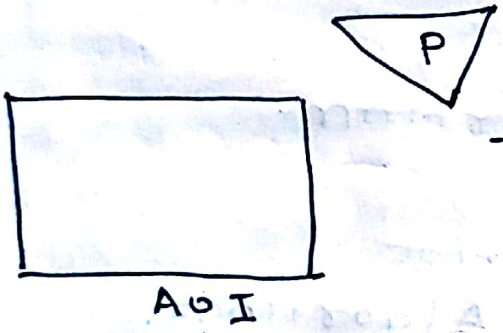
i) Initially the algorithm treats entire area of the projection to be area of interest & finds out the relationship of each sorted polygon with respect to area of

interest & if it couldnot make the decisi-on based on the relationship derived then entire area recursively sub divided into four equal parts of rectongular region and it continues to make the decision.

Four types of relationship wrt Area of interest

i) Disjoint polygon:- When a particular polygon is completely out of the AoI (Area of interest)

Ex:-

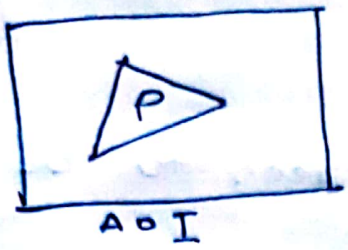


→ P is disjoint wrt AoI (Because it is completely out of AoI)

ii) Contained polygon :-

When a polygon is completely inside the area of interest (AoI).

Ex :-

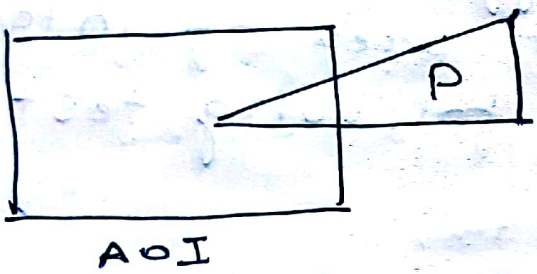


→ P is contained wrt AOI

3) Intersecting polygon:-

When a polygon part of polygon is inside & a part of polygon is outside.

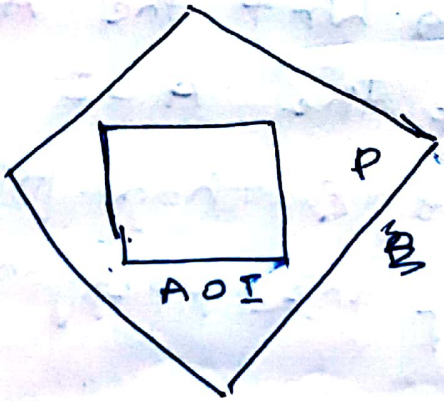
Ex:-



→ P is intersecting wrt area of interest

4) Surrounding polygon:-

When a polygon completely surrounds the area of interest.



→ P is surrounding area of interest

Algorithm :-

*) i) Initialize entire screen to be Area of Interest.

ii) Create a list of polygons sorted in some order based on the z co-ordinate value wrt Area of interest.

iii) For each polygon in the list derive the relationship wrt area of interest.

iv) Perform the following test to make the decision.

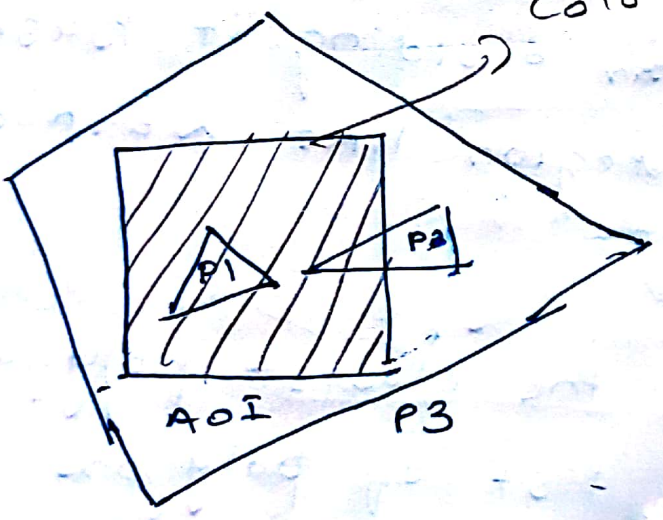
test 1 :- If all the polygon in the list are disjoint wrt AOI then colour the AOI with ~~blue~~ background color.

test 2 :- If there is single contained or intersecting polygon wrt the AOI then first ~~colour~~ color the AOI with background color. & then fill the area aquired by the polygon with polygon color.

Test 3 :- If there is a single surrounding polygon then color the area of interest with polygon color.

Test 4 :- If there are multiple polygons in the AOI & if there is single surrounding polygon closer to the viewpoint then color the AOI with surrounding polygon color :

Ex :-



Color of P3

P1 = 0.75 → Contained
P2 = 0.5 → Intersecting
P3 = 0.25 → Surrounding

P1 = 0.25
P2 = 0.5
P3 = 0.75

test 5 :- If AOI is single pixel P(x,y) and if all the above test fails then the area is ~~cutted~~ coloured with polygon color that is closest to viewer

point

5) ~~Def~~

v) If all the test fails then recursively divide ~~int~~ AoI into 4 equal rectangular part & goto step2.

Weiler Atherton Algorithm :-

* The Algorithm divides the area of projection based on the boundary of particular polygon called as clip polygon rather than dividing it based on rectangular region like Warnock's Algorithm.

* Clip polygon is the closest polygon that comes in the list of polygons.

Conceptual steps :-

1) * Sort the list of polygon based on the decreasing value of z co-ordinate.

2)* Select the first polygon from the list to be the clipped polygon.

3)* With respect to clip polygon subdivide the list into in list & out list even considering the clip polygon itself.

inlist → list of polygon inside clip polygon.

outlist → list of polygon outside clip polygon.

4)* In the inlist if any polygon p is farther from the clip polygon then delete those polygons as they are hidden.

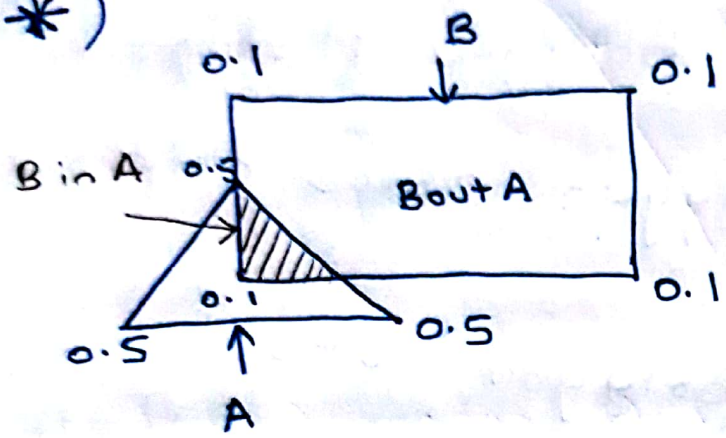
5)* If any polygon in the inlist is closer than clip polygon then subdivide the inlist for each closer polygons recursively.

6)* If inlist has only the clip polygon then display it.

7)* Apply the steps recursively for the outlist.

Weiler Atherton Algorithm

*)



Polylist

A	B

Subdivide (clippolygon, polylist)

Subdivide (A, {B})

inlist | outlist

{A, ~~B in A~~}

↓

display A

{ B out A }

↓

Subdivide (B out A, { })

inlist | outlist

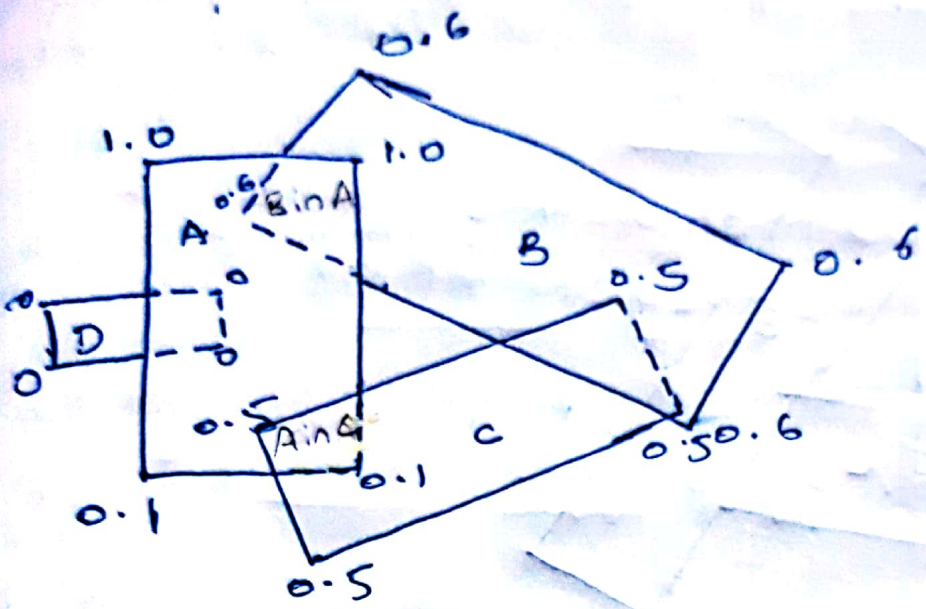
{B out A}

↓

display B out A

{ }

*)



polylist

| A | B | C | D |

subdivide(A, { B, C, D })

inlist | outlist

{ A, ~~BinA~~, CinA, ~~DinA~~ }

(not behind A)

behind A → So strike out

subdivide(CinA, { A })

dont take A as clip polygon as it is CP of previous subdivide

inlist | outlist

~~inlist~~

{ CinA, ~~AinC~~ }

display CinA

~~outlist~~

{ AoutC }

subdivide(AoutC, { })

inlist | outlist

{ AoutC }

display AoutC

{ }

{ BoutA, CoutA, DoutA }

subdivide(BoutA, { CoutA, DoutA })

inlist | outlist

{ BoutA, ~~CoutA~~, ~~inBoutA~~ }

display BoutA

{ CoutA, DoutA, BoutA }

I → CoutA, DoutA, BoutA

J → DoutA, BoutA

subdivide(I, { J })

inlist | outlist

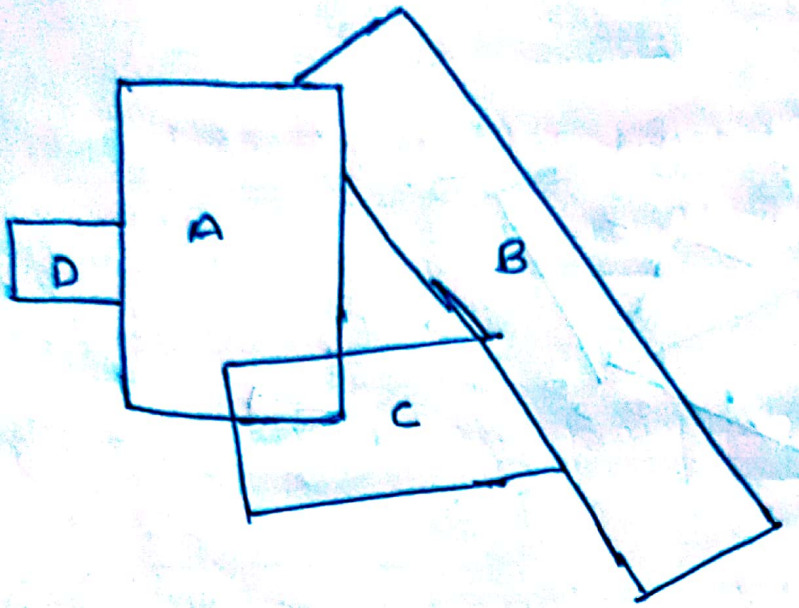
{ I }

display I

{ JoutI }

display JoutI

After :-



Weiler Atherton Algorithm pseudocode

```
WA_ subdivide (clippolygon , polylist[])
{
    if (polylist == NULL)
    {
        return ;
    }
    else
    {
        inlist = NULL;
        outlist = NULL;
    }
    (for each polygon in polylist)
    {
        inlist = list of polygons that is
    inside the clip polygon.
```

outlist = list of polygon that is outside the clip polygon.

```
{
for(each polygon in inlist)
{
  delete polygon that is behind the clip
  polygon;
}
for(each polygon in inlist that is not
     part of clip polygon)
{
  WA_subdivide(polygon, inlist);
}
display polygon in inlist;
  polylist = outlist;
 clippolygon = first polygon(polylist);
   WA_subdivide(clip polygon, polylist);
}
```

polylist = sort polygons based on highest z co-ordinate

clip polygon = first polygon (poly list)

WA - subdivide (clip polygon, polylist);

Unit - III

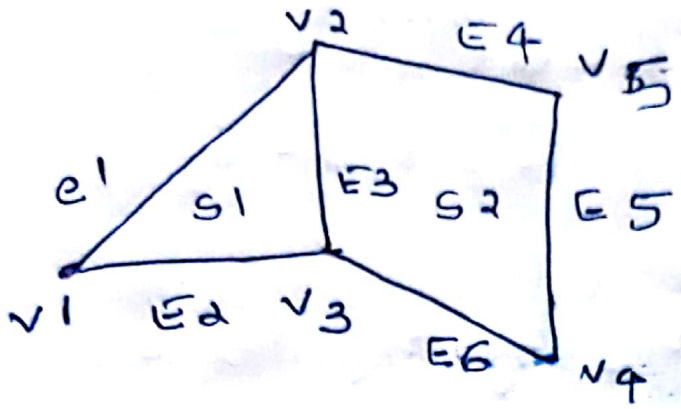
THREE DIMENSIONAL OBJECT & REPRESENTATION

Representation of polygon surfaces :-

* Usually in computer graphics many objects are represented in terms of combination of the polygon surfaces.
* The reason for this is the polygon surfaces are easy to render on the screen since the polygon surface uses the linear equation of the form ~~ax + by +~~

$$Ax + By + Cz + D = 0$$

which is easy to process in terms of mathematical operation.



To represent a polygon there are 3 geometr-
-ic tables are :-

i) Vertex Table
ii) Edge Table
iii) Polygon - surface Table.

i) Vertex Table :-

An vertex Table for a polygon consist of the collection of ~~#~~ vertices as index & within each index it holds x, y, z co-ordinate values corresponding to a particular vertex.

3.

Vertex Table	
V1	x1 y1 z1
V2	x2 y2 z2
V3	x3 y3 z3
V4	x4 y4 z4
V5	x5 y5 z5

* Edge Table :-

It is basically pointer to the vertex table which points to the vertices that defines the particular edge. Like vertex table edge table maintains information pertaining to each edge in a seperate index value.

Edge table	
E1	V1, V2
E2	V1, V3
E3	V2, V3
E4	V2, V5
E5	V4, V5
E6	V3, V4

* Polygon surface table :-

This geometric table will define the edges corresponding to a particular polygon surface. The table consists of a number of indices & each index in the table will hold the information pertaining to particular surface in terms of the edges defining it. This table is basically pointer pointing to the edge table.

Polygon surface table	
S1	E1, E2, E3
S2	E3, E4, E5, E6

Plane equation for the polygon :-

A plane equation is represented as

$$Ax + By + Cz + D = 0$$

Use 3 vertices in clockwise or anti clockwise direction, V1, V2, V3

$$V1 = Ax1 + By1 + Cz1 + D = 0$$

$$V2 = Ax2 + By2 + Cz2 + D = 0$$

$$V3 = Ax3 + By3 + Cz3 + D = 0$$

$$V1 \Rightarrow (A/D)x1 + (B/D)y1 + (C/D)z1 = -1$$

$$V2 = (A/D)x2 + (B/D)y2 + (C/D)z2 = -1$$

$$V3 = (A/D)x3 + (B/D)y3 + (C/D)z3 = -1$$

According to cramers rule :-

$$A = \begin{vmatrix} 1 & y1 & z1 \\ 1 & y2 & z2 \\ 1 & y3 & z3 \end{vmatrix}$$

$$B = \begin{vmatrix} x1 & 1 & z1 \\ x2 & 1 & z2 \\ x3 & 1 & z3 \end{vmatrix}$$

$$C = \begin{vmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{vmatrix}$$

$$D = \begin{vmatrix} x1 & y1 & z1 \\ x2 & y2 & z2 \\ x3 & y3 & z3 \end{vmatrix}$$

$A = y_1(z_2 - z_3) - y_2(z_1 - z_3) + y_3(z_1 - z_2)$

$B = z_1(x_2 - x_3) - z_2(x_1 - x_3) + z_3(x_1 - x_2)$

$C = x_1(y_2 - y_3) - x_2(y_1 - y_3) + x_3(y_1 - y_2)$

$$D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

$D = x_1(y_2z_3 - y_3z_2) - y_1(x_2z_3 - x_3z_2) + z_1(x_2y_3 - y_2x_3)$

For any plane if $A = 1, B = 0, C = 0$ & $D = -1$ then the plane is in front of view plane else it is backside.

For any point (x, y, z) if $Ax + By + Cz + D < 0$ then point is inside the plane.

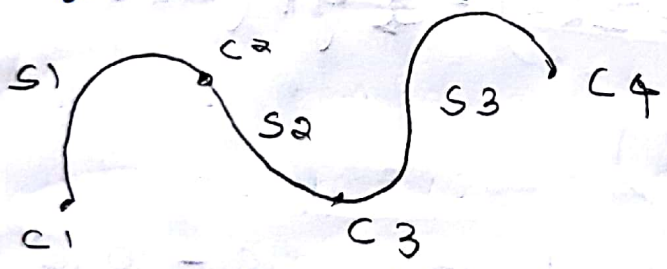
For any point (x, y, z) if $Ax + By + Cz + D = 0$ then point is on the edge of the plane.

for any point (x, y, z) if $Ax + By + Cz + D > 0$ then point is outside the plane.

imp

Spline Representation :-

* Spline in drafting technology is a flexible strip that is used to produce a curve shape through a designated set of points.
* Spline in CG basically represent a composite curve that is being generated by fitting a polynomial expression to each section of the curve that satisfyes specific continuity conditions.



imp

Types of splines

* Based on the way the spline intersects with the control points there are 2 types of splines:-

i) Interpolation spline.

ii) Approximation Spline.

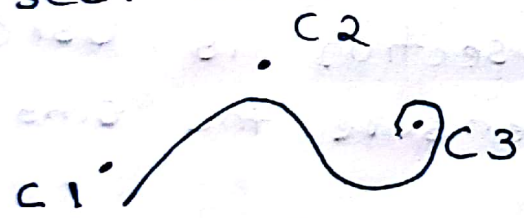
* In interpolation spline a curve will intersect all the designated control points.

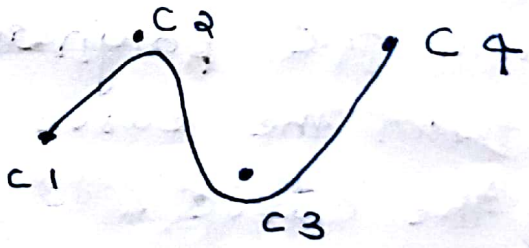
*

* Here a polynomial equation is fitted such a way that it ensures the intersection with each control points.

ii) Approximation spline :-

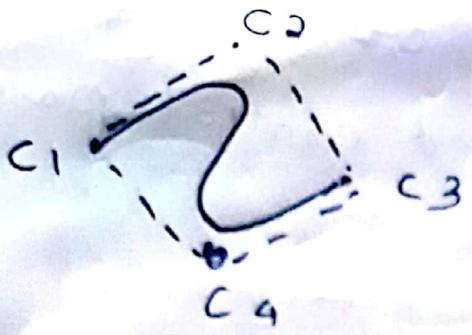
In approximation spline curve is generated such a way that it need not necessarily intersect each control point.





Convex hull :-

* It is a convex polygon formed out of the designated set of control points.



Continuity Conditions :-

* Condition that ensures the smoother transition of curve from one section to another section.

2 types :-

* Parametric Continuity.
* Geometric continuity.

* **Parametric Continuity** :- Here the parametric polynomial section is used between the curve to ensure the smoothness of the curve.

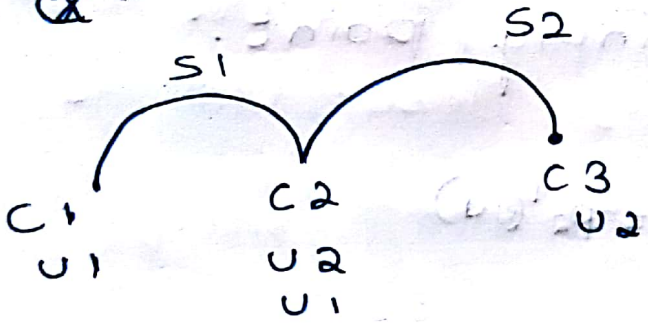
* A parametric equation is fixed to each coordinate values at the point which joins two section. In the form

$$x = x(u)$$
$$y = y(u)$$
$$z = z(u)$$

3 types of parametric continuity -

1. zero order continuity → called as C° continuity
2. First order continuity.
3. second order continuity.

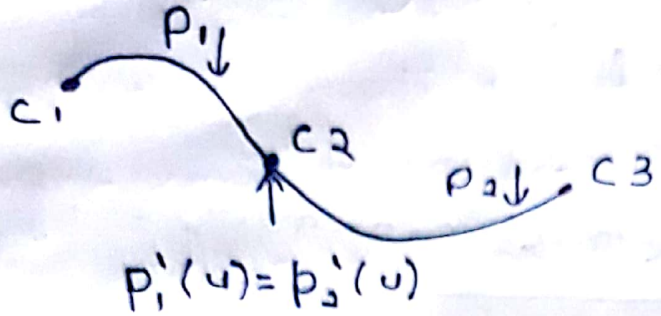
* Zero order → is called as C° continuity in this continuity the u2 value of section 1 is equal to u1 value of section 2.



* First order parametric continuity :-
In first order parametric continuity the first order parametric derivative for the 2 sections of the curve is equal at their joining point.

$P_1(u) \Rightarrow P_1'(u)$

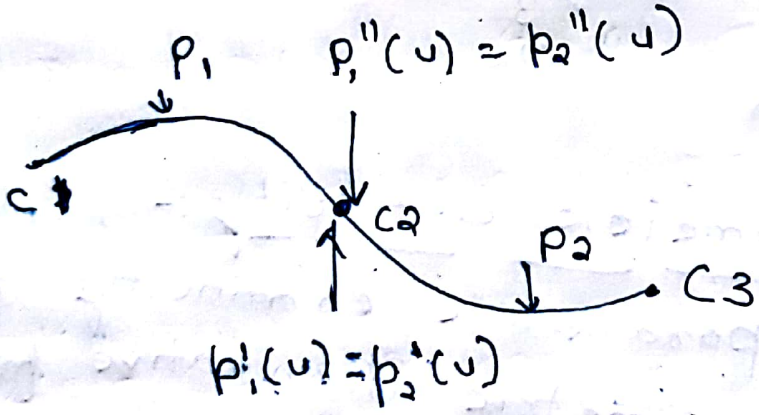
$P_2(u) \Rightarrow P_2'(u)$



→ Also called as C¹ continuity

* Second order parametric continuity :-

In 2nd order parametric continuity the first order parametric derivative as well as 2nd order parametric derivative for the 2 sections of the curve has to be same at the joining point.



Geometric continuity :-

This is an alternate method for the parametric continuity where the geometric continuity doesnot impose the parametric

derivative to be necessarily same but they impose the condition of the parametric derivatives to be proportional.

Like parametric continuity even geometric continuity has 3 order of continuities.

* Zero order geometric continuity :-

It is described as G^0 continuity and is very similar to C^0 continuity that requires the parameteric section of the point joining 2 sections of the curves to be same i.e the parametric value of u2 for the first curve is directly equal to the parametric value u1 of the 2nd curve.

* First order geometric continuity :-

In the first order geometric continuity it requires first order parametric derivative of 1 curve section to be proportional to the first order parametric derivative of another curve section.

if $p_i'(u)$ is 1st order parametric

derivative of $P_1(u)$ & $P_2'(u)$ & is the 1st order derivative of 2nd section then if we get result in which $P_2'(u) = \frac{1}{P_1'(u)}$

Then type of continuity is called as 1st order geometric continuity.

* It is also described as G^1 continuity.

Ex:- $2x+2 \quad \frac{1}{2x+2}$

* 2nd order geometric continuity :-

It requires both the first order parametric derivative as well as 2nd order parametric derivative to be proportional to each other. It is also described as G^2 continuity.

Ex :- $P_1''(u) = 2 \quad \frac{1}{2}$

Till last write for exam

Bezier curves and Surfaces :-

* Bezier's line is one of the method for producing the curved surfaces called as bezier's curve.

* Bezier curve was originally invented by piere bezier for the design of the body of the renault automobile.

* A Beziers lines are basically one of the easiest splines to be generated, hence it is used widely in different areas of computer graphics.

Properties of Bezier Curves :-

* A Bezier curve always intersects with the first and last control points.

* A Bezier curve with (n+1) control points has polynomial section of degree n.

* A shape of the bezier curve will always lie inside the convex hull.

* The slope of the bezier curve at the first section will always be towards a line joining first 2 control sections & the slope of the bezier curve at the last section will always be towards the line joining last 2 control points.

* Even though the Bezier curve lies within the convex hull, the curve could not be controlled or modified locally at specified control point due to the property of its polynomial section.

Polynomial Section for Bezier Curve :-

$$p(u) = \sum_{k=0}^{n} p_k \cdot BEZ_{k,n}(u) \qquad 0 \le u \le 1$$

$p_k \rightarrow$ is array co-ordinate value to which a section of Bezier curve needs to be fitted.

$p_k = \{ x_k, y_k, z_k \}$

$BEZ_{k,n}(u) \rightarrow$ Blending function.

$BEZ_{k,n}(u) = C(n,k)\, u^k \cdot (1-u)^{n-k}$

$C(n,k) = \dfrac{n!}{k!\,(n-k)!}$

Derivation of Cubic Bezier Spline :-

* Bezier spline having the polynomial section with degree $(n) = 3$.

Control points $(n+1) = 3+1 = 4$

$$p(u) = \sum_{k=0}^{3} p_k \, BEZ_{k,3}(u) \quad ; \quad 0 \le u \le 1$$

$BEZ_{0,3}(u)$

$BEZ_{1,3}(u)$

$BEZ_{2,3}(u)$

$BEZ_{3,3}(u)$

$$BEZ_{0,3}(u) = C(3,0)\, u^0 (1-u)^3$$

$$= \frac{3!}{0!(3-0)!} \cdot 1 \, (1-u)^3 = (1-u)^3$$

$$BEZ_{0,3}(u) = (1-u)^3$$

$$BEZ_{1,3}(u) = C(3,1)\, u^1 (1-u)^2$$

$$= \frac{3!}{1!(3-1)!} \times u (1-u)^2$$

$$= 3u(1-u)^2$$

$$BEZ_{2,3}(u) = C(3,2)\, u^2 (1-u)^1$$

$$= \frac{3!}{2!(3-2)!} \times u^2 (1-u)$$

$$= 3u^2(1-u)$$

$$BEZ_{3,3}(u) = C(3,3)\, u^3 (1-u)^0$$

$$= \frac{3!}{3!(0)!} \times u^3 = u^3$$

$BEZ_{0,3}(u) = (1-u)^3$

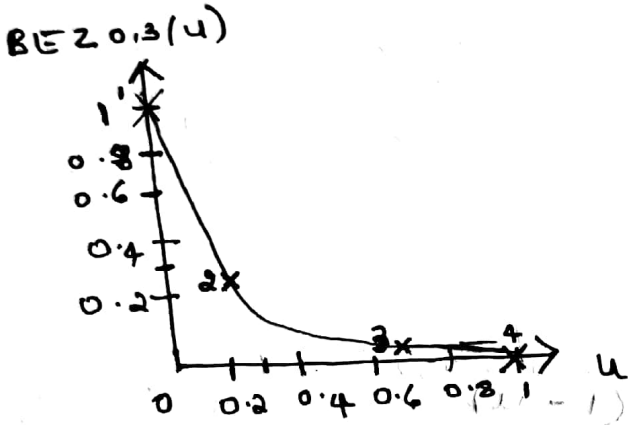
$BEZ_{1,3}(u) = 3u(1-u)^2$

$BEZ_{2,3}(u) = 3u^2(1-u)$

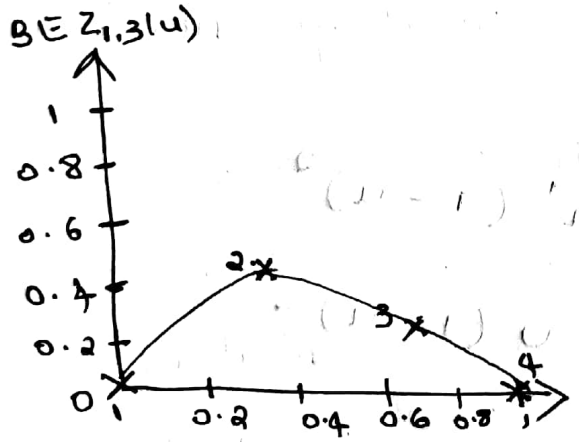
$BEZ_{3,3}(u) = u^3$

$u = 0/3, 1/3, 2/3, 3/3 \longrightarrow 0, 0.33, 0.666, 1$

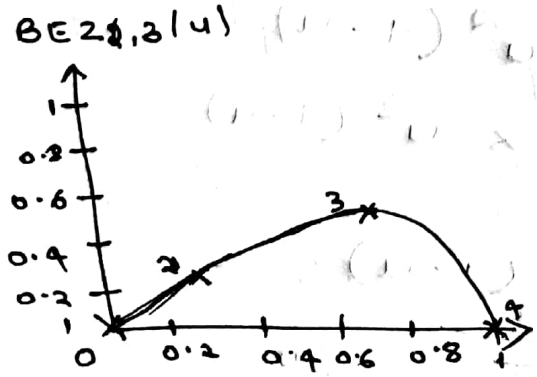
for $BEZ_{0,3}(u)$



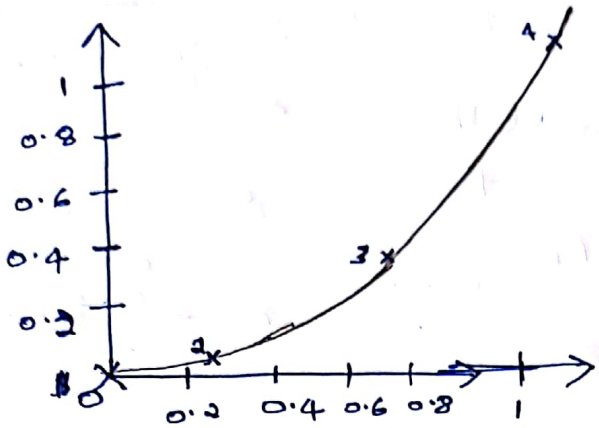
for $BEZ_{1,3}(u)$



for $BEZ_{2,3}(u)$



for BEZ 3,3 (u)



imp

B Spline Curve :-

* Advantages of B-Spline Curve over bezier curve:-

* The degree of the polynomial equation in B-spline curve is independent of the number of control points with some condition.

* The B spline curve can be controlled locally over the control point.

Drawback:-

* It is difficult to understand and implement.

imp

Polynomial Equation for Bspline Curve

$$p(u) = \sum_{k=0}^{n} P_k B_{k,d}(u) \qquad u_{min} \le u \le u_{max}$$

$$2 \le d \le n+1$$

$B_{k,d}(u)$ → Blending Function.

d → degree of the polynomial.

$n+1$ → no. of control points.

* Each of the blending function has sub intervals that is defined by the value of knot vector u.

* The Expansion for $B_{k,d}(u)$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k}$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

$$B_{k,1}(u) = \begin{cases} 1 & u_k \le u < u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

Construction of knot vector :-

* Knot vector is basically an array that is constructed for selecting the range of u value that ranges from minimum to maximum value. Each value in the knot vector should be greater or equal than ith previous value & less than & equal to next value.

* If there are n+1 control point for the curve & if degree of the polynomial chosen is d which is $2 \le d \le n+1$, then knot vector will have n+d+1 values inside it.

i) Uniform periodic B spline :-

* In this type of knot vector there will be an uniform distribution of values between any two elements of u.

Ex :- $n = 3,\ d = 3$

$n + d + 1 = 7$

$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6\} = \{-1.0, -0.5, 0, 0.5, 1.0, 1.5, 2.0\}$

$= \{1, 2, 3, 4, 5, 6, 7\}$

ii) Uniform open B spline :-

~~$u_j = \begin{cases} 0, & \text{for first } d \text{ values} \\ j - d + 1 \end{cases}$~~

$$u_j = \begin{cases} 0 & \text{for first } d \text{ values} \\ j - d + 1 & d \le j \le n \\ n - d + 2 & j \ge n \end{cases}$$

$n = 3,\ d = 3$

$n + d + 1 = 7$ values

$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6\} = \{0, 0, 0, 1, 2, 2, 2\}$

iii) Non uniform B spline :-

$n + d + 1 = 7$

$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6\} = \{2, 5, 6, 6, 7, 7, 10\}$

random values but it should be in increasing order.

B Spline Curve

Design of quadratic uniform B-spline curve

$(n+1 = 4)$ ↓ 4 control points

$n = 3, d = 3$

$n + d + 1 = 7$

$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6\} = \{0, 1, 2, 3, 4, 5, 6\}$,

$u_k \le u \le u_{k+1}$

$$p(u) = \sum_{k=0}^{3} p_k B_{k,3}(u)$$

$$B_{k,d} = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

$$B_{i,1}(u) = \begin{cases} 1, & u_k \le u \le u_{k+1} \\ 0, & \text{otherwise.} \end{cases}$$

Blending function :-

$B_{0,3}(u)$
$B_{1,3}(u)$
$B_{2,3}(u)$
$B_{3,3}(u)$

$u_0 = 0$, $u_2 = 2$ } from knot vector

$$B_{0,3}(u) = \frac{u - u_0}{u_2 - u_0} B_{0,2}(u) + \frac{u_3 - u}{u_3 - u_1} B_{1,2}(u)$$

$$= \frac{u - 0}{2 - 0}\left[\frac{u - u_0}{u_1 - u_0} B_{0,1}(u) + \frac{u_2 - u}{u_2 - u_1} B_{1,1}(u)\right]$$

$$+ \frac{3 - u}{3 - 1}\left[\frac{u - u_1}{u_2 - u_1} B_{1,1}(u) + \frac{u_3 - u}{u_3 - u_2} B_{2,1}(u)\right]$$

d-1=1

when d=1 it is end

$$= \frac{u}{2}\left[\frac{u}{1} B_{0,1}(u) + \frac{(2-u)}{1} B_{1,1}(u)\right] + \frac{3-u}{2}\left[(u-1) B_{1,1}(u) + (3-u) B_{2,1}(u)\right]$$

$$\left[\begin{array}{l} u_k \le u \le u_{k+1} \\ 0 \le u \le 1 \\ 1 \le u \le 2 \\ 2 \le u \le 3 \end{array}\right\} \text{For } B_{0,3}(u) \quad]$$

for $0 \le u \le 1$

$$\frac{u}{2}\left[u \cdot 1 + (2-u) \cdot 0\right] + \frac{3-u}{2}\left[(u-1)\cdot 0 + (3-u)\cdot 0\right]$$

for $1 \le u \le 2$

$$\frac{u}{2}\left[u \cdot 0 + (2-u)\cdot 1\right] + \frac{3-u}{2}\left[(u-1)\cdot 1 + (3-u)\cdot 0\right]$$

for $2 \le u \le 3$

$$\frac{u}{2}\left[u \cdot 0 + (2-u)\cdot 0\right] + \frac{3-u}{2}\left[(u-1)\cdot 0 + (3-u)\cdot 1\right]$$

for $0 \le u \le 1 \quad (k=0)$

$$\frac{u}{2}[u]$$

for $1 \le u \le 2 \quad (k=1)$

$$\frac{u}{2}[(2-u)] + \frac{3-u}{2}[(u-1)]$$

$$= \frac{u(2-u) + (3-u)(u-1)}{2}$$

for $2 \le u \le 3$ $(k=2)$,

$$\frac{3-u}{2}\left[(3-u)\right]$$

$$\frac{(3-u)^2}{2}$$

$$B_{0,3}(u) = \begin{cases} \frac{u^2}{2}, & 0 \le u \le 1 \\ \frac{u(2-u) + (3-u)(u-1)}{2}, & 1 \le u \le 2 \\ \frac{(3-u)^2}{2}, & 2 \le u \le 3 \end{cases}$$

(below steps only work if we use uniform periodic b-sp else we have to solve everything)

$B_{1,3}(u)$ → subtract with diff, 1 & 0 between

$$B_{1,3}(u) = \begin{cases} \frac{(u-1)^2}{2}, & 1 \le u \le 2 \\ \frac{(u-1)(3-u) + (4-u)(u-2)}{2}, & 2 \le u \le 3 \\ \frac{(4-u)^2}{2}, & 3 \le u \le 4 \end{cases}$$

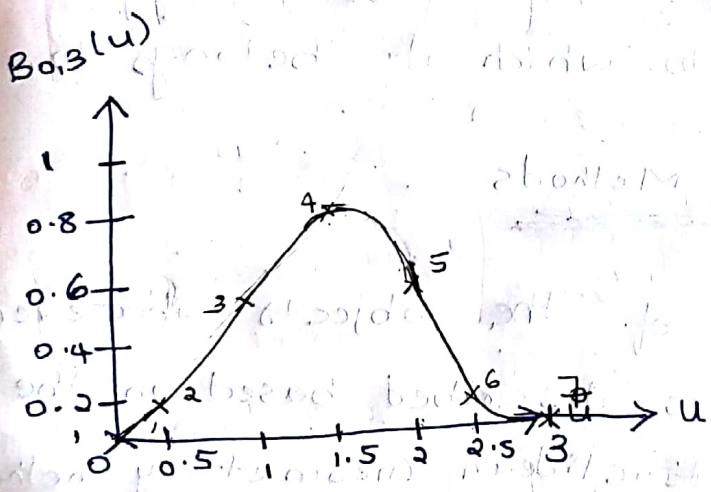
$$B_{2,3}(u) = \begin{cases} \frac{(u-2)^2}{2}, & 2 \le u \le 3 \\ \frac{(u-2)(4-u) + (5-u)(u-3)}{2}, & 3 \le u \le 4 \\ \frac{(5-u)^2}{2}, & 4 \le u \le 5 \end{cases}$$

$$B_{3,3}(u) = \begin{cases} \frac{(u-3)^2}{2}, & 3 \le u \le 4 \\ \frac{(u-3)(5-u)+(6-u)(u-4)}{2}, & 4 \le u \le 5 \\ \frac{(6-u)^2}{2}, & 5 \le u \le 6 \end{cases}$$

Plotting :-

or $B_{0,3}(u)$

$$B_{0,3}(u) = \begin{cases} \frac{u^2}{2}, & 0 \le u \le 1 \\ \frac{u(2-u)+(3-u)(u-1)}{2}, & 1 \le u \le 2 \\ \frac{(3-u)^2}{2}, & 2 \le u \le 3 \end{cases}$$



Properties of B-spline curve :-

* A polynomial curve of B spline has degree d-1 & continuity of C^{d-2} over a range of u.
* A polynomial curve with n+1 control point has a blending functions of n+1 starting from k=0 & ending with k=n.

* Each blending function is defined over d number of sub interval, with each sub interval has starting value equivalent to k & an ending value equivalent to k+1 chosen from knot vector u.

* Total no of elements for the knot vector is equals to n+d+1 value which is arranged in ~~equal~~ higher order such that the previous value is less than or equals to next value.

* Each section of curve is controlled by d no of control points.

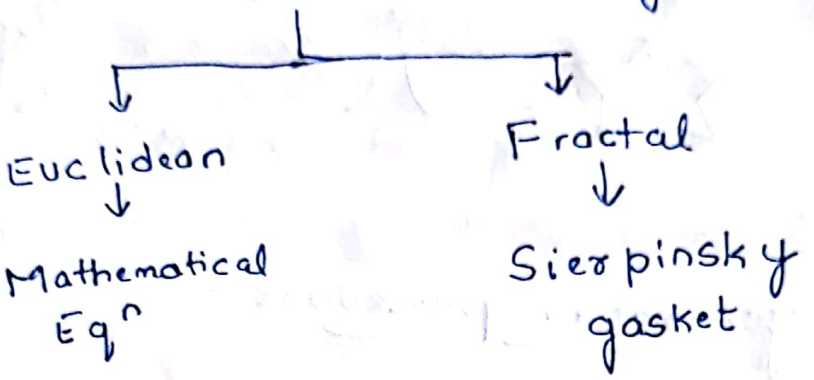
* Any one control point can affect only ~~d~~ the d sections to which it belongs to.

Fractal Geometry Methods

Generally so far shapes of the objects whatever we have studied is described based on the method called as Euclidean Geometery methods. Where shape of the objects is described ~~as~~ in terms of Mathematical Equations.

Fractal Geometry methods are those methods which are used to describe the shape of those objects which has an irregular shape. In this type of geometry method the procedures are used to described

The shape of the object rather than the mathematical equations. This are used to represent the objects that occurs naturally in real time.



Properties of Fractal Geometry method :-

Every fractal geometry methods basically has 2 important properties.

i) Infinite details

ii) Self similarity.

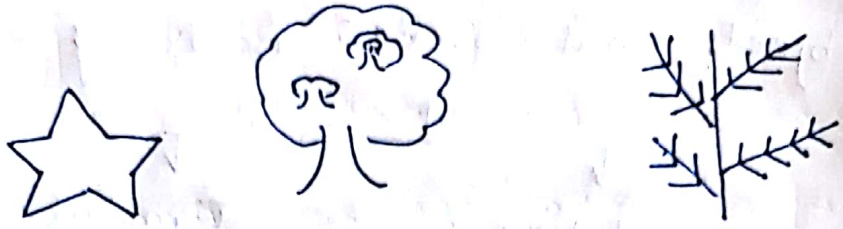
* Infinite details :- Is one such property exhibited by the fractal geometry method where in every fractal geometry method has to exhibit infinite amount of details as the object is zoomed closer & closer.

For Ex :- When the mountain object is Zoomed it has to reveal the diff object like rocks & other things lying within the mountain.

* Self similarity :- This property of the fractal object defines similarity within certain part of object & it depends

on the method chosen for the fractal object representation.

Ex:-



Fractal generation procedure:-

In simple a fractal object is generated recursively by applying specified transformat function to a point or to a set of point within a region of space.

That is if $P_0 = (x_0, y_0, z_0)$ represents a single point of the initial fractal object then for each recursion the transformation function generates the next level of fractal object that reveals next level of detail. The successive level of transformation is mathematical specified as

$$P_1 = F(P_0), \quad P_2 = F(P_1), \quad P_3 = F(P_2)$$

$$\ldots\ldots P_n = F(P_{n-1})$$

where,

F represents some kind of transformation based on the procedure & $P_0, P_1,$ upto P_{n-1} represent the fractal objects formed out in the previous

recursion.

Classification of fractals :-

Are classified into 3 different types based on the method or the procedure that is used to generate the fractal object.

i) Self Similar Fractals :-

This are constructed by applying some scaled down parameters to the part of entire object. The procedure in this type of fractal is some transformation function that will represent scaled down object at certain parts having the similar shape of the object.

Some of the good example for this are :-

Trees, Sierpinsky gasket etc.

ii) Self Affine fractal :-

Are the fractal object formed by applying different scaling transformations by using different scaling parameters for Sx, Sy, S_z at different co-ordinate directions.

Ex for this kind of fractals :-

Clouds, Water terrance etc.

ii) Invariant Fractals :-

In this type of fractal generation method a non linear transformation functions is used in which case transformation function will not be uniform across the different recursive procedure.

As a standard this type of fractals includes the transformation such as self squaring fractals which involves generation of fractals through squaring function.

One of good example for this is a fractal called as Mandle Brot.

Ex:- $P1 = F(P0)$

$P2 = F^2(P1)$

$P3 = (F^2)^2(P2) = F^4 P2$

or

$P1 = F(P0)$

$P2 = F1/2(P0)$

$P3 = F1/4(P1)$

Illumination and Shading :-

Illumination Model :-

An illumination model is basically mathematical model that describes the color of object surface with respect to the intensity of light & various other external factors.

It basically describes the color of the surface wrt the intensity of the surface which is used to produce a realistic view.

Shading model :-

* Shading model is a model that describes the color of surface by considering a particular illumination model to be applied over the surface of the object to derive the realistic view.
* Shading model will describe types of illumination that ~~does~~ needs to be used considering the surface properties & other external factors.

imp
Different illumination Models :-

imp
* Ambient light illumination Model :-

→ Natural source of Light that is present in the atmosphere.

→ Ambient light illumination model describes the ~~iterati~~ ~~intensity~~ intensity of the object surface when it is illuminated by an

ambient light uniformly.

The intensity is given by:

$$I = I_a \cdot k_a$$

Where I = ~~iten~~ intensity of ~~the~~ an object surface.
I_a = intensity of ambient light
k_a = object reflection co-efficient for Ambient light.

$k_a = 0$ for matt surface
$k_a = 1$ for mirrored surface.

2) Diffuse Reflection :-

* This illumination model describes intensity of an object surface when an object surface is uniformly illuminated by a point source light (torch). This model describes the intensity with respect to the angle of light source that falls on the object surface as well as with respect to the distance of the light source from the object.

* Different Diffuse Reflection illumination models are :-

i) Lambertian Reflection.
ii) Light source attenuation.
iii) Colored Light source model.

i) Lambertian Reflection :- This model of diffuse reflection is used to define the ~~item~~ intensity of the object with matt surfaces that appears equally bright in all directions when it is illuminated by a point source of light.

* According to this technique intensity of the surface mainly depends on 2 factors :-

i) Direction of Light source. wrt the surface normal ($\bar{L}$). (describes front side of the surface)

ii) Angle of Light source reflection wrt to the surface normal (θ)

$$I = I_p \cdot \cos\theta$$

I = overall intensity ~~of point source~~ light

I_p = intensity of point source light

$\cos\theta$ = Angle between $\bar{N}$ & $\bar{L}$

$$\cos\theta = \bar{N} \cdot \bar{L}$$

$$I = I_p \cdot (\bar{N} \cdot \bar{L})$$

Considering the surface reflection co-efficient it is given by

$$I = I_p K_d (\bar{N} \cdot \bar{L})$$

K_d = diffuse Reflection Coefficient.

If $K_d = 1$, for complete mirrored surface.

$K_d = 0$, for perfect dull (matt) surface.

The overall intensity of object surface is now given by

$$I = I_a K_a + I_p K_d \cos\theta$$

Where

$$\cos\theta = \overline{N} \cdot \overline{L}$$